```
#ifndef __MTHREAD_MTHREAD_H__
#define __MTHREAD_MTHREAD_H__
#ifdef __cplusplus
extern "C"
{
#endif
  /* Types */
  typedef volatile unsigned int mthread_tst_t;

  struct mthread_s;
  typedef struct mthread_s* mthread_t;

  struct mthread_attr_s;
  typedef struct mthread_attr_s mthread_attr_t;

  struct mthread_mutex_s;
  typedef struct mthread_mutex_s mthread_mutex_t;

  struct mthread_mutexattr_s;
  typedef struct mthread_mutexattr_s mthread_mutexattr_t;

  struct mthread_cond_s;
  typedef struct mthread_cond_s mthread_cond_t;

  struct mthread_condattr_s;
  typedef struct mthread_condattr_s mthread_condattr_t;

  typedef unsigned int mthread_key_t;

  struct mthread_once_s;
  typedef struct mthread_once_s mthread_once_t;

  struct mthread_sem_s;
  typedef struct mthread_sem_s mthread_sem_t;

  /* Function for handling threads.  */

  /* Create a thread with given attributes ATTR (or default attributes
     if ATTR is NULL), and call function START_ROUTINE with given
     arguments ARG.  */
  extern int mthread_create (mthread_t * __threadp,
                             const mthread_attr_t * __attr,
                             void *(*__start_routine) (void *), void *__arg);

  /* Obtain the identifier of the current thread.  */
  extern mthread_t mthread_self (void);

  /* Compare two thread identifiers.  */
  extern int mthread_equal (mthread_t __thread1, mthread_t __thread2);

  /* Terminate calling thread.  */
  extern void mthread_exit (void *__retval);

  /* Make calling thread wait for termination of the thread TH.  The
```

```
   exit status of the thread is stored in *THREAD_RETURN, if THREAD_RETURN
   is not NULL.  */
extern int mthread_join (mthread_t __th, void **__thread_return);

/* Functions for mutex handling.  */

/* Initialize MUTEX using attributes in *MUTEX_ATTR, or use the
   default values if later is NULL.  */
extern int mthread_mutex_init (mthread_mutex_t * __mutex,
                               const mthread_mutexattr_t * __mutex_attr);

/* Destroy MUTEX.  */
extern int mthread_mutex_destroy (mthread_mutex_t * __mutex);

/* Try to lock MUTEX.  */
extern int mthread_mutex_trylock (mthread_mutex_t * __mutex);

/* Wait until lock for MUTEX becomes available and lock it.  */
extern int mthread_mutex_lock (mthread_mutex_t * __mutex);

/* Unlock MUTEX.  */
extern int mthread_mutex_unlock (mthread_mutex_t * __mutex);

/* Functions for handling conditional variables.  */

/* Initialize condition variable COND using attributes ATTR, or use
   the default values if later is NULL.  */
extern int mthread_cond_init (mthread_cond_t * __cond,
                              const mthread_condattr_t * __cond_attr);

/* Destroy condition variable COND.  */
extern int mthread_cond_destroy (mthread_cond_t * __cond);

/* Wake up one thread waiting for condition variable COND.  */
extern int mthread_cond_signal (mthread_cond_t * __cond);

/* Wake up all threads waiting for condition variables COND.  */
extern int mthread_cond_broadcast (mthread_cond_t * __cond);

/* Wait for condition variable COND to be signaled or broadcast.
   MUTEX is assumed to be locked before.  */
extern int mthread_cond_wait (mthread_cond_t * __cond,
                              mthread_mutex_t * __mutex);

/* Functions for handling thread-specific data.  */

/* Create a key value identifying a location in the thread-specific
   data area.  Each thread maintains a distinct thread-specific data
   area.  DESTR_FUNCTION, if non-NULL, is called with the value
   associated to that key when the key is destroyed.
   DESTR_FUNCTION is not called if the value associated is NULL when
   the key is destroyed.  */
extern int mthread_key_create (mthread_key_t * __key,
                               void (*__destr_function) (void *));
```

```
  /* Destroy KEY.  */
  extern int mthread_key_delete (mthread_key_t __key);

  /* Store POINTER in the thread-specific data slot identified by KEY. */
  extern int mthread_setspecific (mthread_key_t __key, const void *__pointer);

  /* Return current value of the thread-specific data slot identified by KEY.  */
  extern void *mthread_getspecific (mthread_key_t __key);


  /* Functions for handling initialization.  */

  /* Guarantee that the initialization function INIT_ROUTINE will be called
     only once, even if mthread_once is executed several times with the
     same ONCE_CONTROL argument. ONCE_CONTROL must point to a static or
     extern variable initialized to MTHREAD_ONCE_INIT.

     The initialization functions might throw exception which is why
     this function is not marked with .  */
  extern int mthread_once (mthread_once_t * __once_control,
                           void (*__init_routine) (void));

  /* Functions for handling semaphore.  */

  extern int mthread_sem_init (mthread_sem_t * sem, unsigned int value);
  extern int mthread_sem_wait (mthread_sem_t * sem);          /* P(sem), wait(sem) */
  extern int mthread_sem_post (mthread_sem_t * sem);          /* V(sem), signal(sem) */

  extern int mthread_sem_getvalue (mthread_sem_t * sem, int *sval);
  extern int mthread_sem_trywait (mthread_sem_t * sem);

  extern int mthread_sem_destroy (mthread_sem_t * sem);       /* undo sem_init() */

  extern void mthread_yield();

#ifdef __cplusplus
}
#endif
#endif
```